Identity Armour: User Controlled Browser Security

Ross Copeland and Drew Davidson(✉)

EECS Department, ITTC University of Kansas, Lawrence, KS, USA
{rcopeland,drewdavidson}@ku.edu

Abstract. As dynamic technologies are deployed to make the web more responsive and feature-rich, the abuse of these capabilities have given rise to emergent privacy and security concerns. At the same time, the prevalence of targeted advertising-driven revenue streams has built an incentive to amass more information about visitors and little incentive to prevent third-party entities from collecting such data. We create a prototype policy enforcement system called Identity Armour that is purely client-side, requiring no cooperation from the site developer. Our system can enforce policies over the functionality of practical JavaScript, including the ability to prevent data that users consider to be sensitive and to prevent functions that the user considers to be prohibited from being executed. We show that Identity Armour is effective at stopping real privacy leakages, and equips users with modern web protections even when first-party developers fail to supply policies themselves.

Keywords: Web security · User privacy · Cross-site scripting

1 Introduction

An alarming trend exists in web security: although Americans are increasingly concerned about the impact of their private online data being misused [2], they are also increasingly certain that their data will be compromised [11]. A particular focus for user privacy and security is in the browser. As browsers are enhanced to support the needs of increasingly feature-rich websites and web applications, complex new capabilities can serve as vectors for attack on user privacy. As website and web applications evolve to use these capabilities, several factors exacerbate the threat of privacy invasion.

Much of the challenge in mitigating attacks via these vectors is in discerning which capabilities are acceptable operations for a user. Furthermore, enhanced browser capabilities are not without legitimate purpose and their appropriateness depends heavily on context. For example, although HTTP cookies make website more convenient and personalized, they can be used to violate a user's privacy by tracking their browsing behavior. Ultimately, the distinction between what information a user is willing to share with a first-party website (and with

N. Park et al. (Eds.): SecureComm 2020, LNICST 335, pp. 474–480, 2020.
https://doi.org/10.1007/978-3-030-63086-7_26

any third-party embedded content) is a personal decision with no one-size-fits-all solution: some users are willing to give up more privacy than others [8,10].

In recognition of the potential danger of dynamic web content, a number of policy enforcement systems, such as Content Security Policy (CSP), have been developed. These mechanisms allow a web developer to mediate the behavior and provenance of active content, most notably client-side JavaScript, by disallowing functionality. CSP can be used as a mitigation against cross-site scripting attacks (XSS), in which an adversary injects content onto a target website in the hope of exfiltrating data from a victim user of the site. While these defenses exist, their use is sparse, and often deployed in a "report-only mode that does not prevent attacks" [9]. Furthermore, the CSP may not be in the best interest of the user.

To combat the threat of data exfiltration through client-side JavaScript, we create Identity Armour, a prototype policy enforcement mechanism that, unlike the above-mentioned protections, does not rely upon the first-party. The contributions of our work are as follows:

- **Custom policy language**: We develop a novel policy language that allows users to choose what information is shared with first-party and third-party entities, mitigate the behavior of active content such as scripts, and identify possible privacy leaks.
- **Identity Armour prototype**: We implement a prototype version of Identity Armour in order to show that our custom policy language can be enforced in practice.
- **Identity Armour evaluation**: We evaluate our Identity Armour prototype, and show that it is effective at preventing real and synthetic exfiltration attacks. We also show that enforcement overhead is comparable to other methods proposed in the literature.

The rest of our paper is structured as follows: Sect. 2 overviews our technical solution, and discusses our implementation of Identity Armour. Section 3 presents our evaluation of Identity Armour. Section 4 discusses related work, and Sect. 5 concludes.

2 Overview

In this section we outline the use and implementation of Identity Armour. As a broad synopsis, Identity Armour is a Firefox XPCOM extension built on top of the JSFlow taint tracking JavaScript interpreter [3]. Identity Armour's enforcement engine follows a combined policy of a user-defined policy paired with a base policy that defines how JavaScript may act on any given webpage. The next section goes into detail as how this policy language functions within the enforcement engine and what JavaScript behavior the policy controls.

2.1 Policy Language

The policy language is whitelisting based and defines JavaScript behavior on a per domain basis. If any code from the webpage does not behave according

to the rules of the defined policy, the enforcement engine stops it from executing. The policy language defines what sources of data are taint tracked, which external JavaScript libraries to load, and which JavaScript functions should be allowed to execute, all at runtime. The user policy is to have a customizable experience on a per webpage basis, and depending on the webpage, the policy can be more restrictive, based on the necessary actions the webpage needs to function correctly. The base policy is used to define actions that are allowed by a webpage that is not defined in the user policy. A webpage not defined in the user policy will still have its JavaScript be restricted by the specifications of the base policy. So that the technologically inexperienced user does not have the burden of implementing these user-defined policies, we implemented a learning aspect in which a user would visit a website and Identity Armour would then create a policy based off of the required actions for the website to perform normally. The system comes with a base policy pre-written, so the user does not have the duty of creating one.

2.2 Implementation Details

We use an extension because they are easily modified and deployed. Additionally, the work of Hedin et al. had already proved JSFlow is an effective taint tracking interpreter [3]. The enforcement engine of Identity Armour uses the dynamic taint tracking of JSFlow to trace individual data flows throughout the website's JavaScript to enforce that no sensitive data leaks. To effectively track data flows through the website's code, we interpose on low-level data propagation operations in JavaScript. Although JSFlow provides an underlying mechanism by which data flow can be tracked throughout execution of a web page (or web application), it does entail some limitations: an older version of Firefox, Firefox 30, must be used as newer browsers disallow implementing a custom JavaScript interpreter as extensions (a core design choice of JSFlow).

2.3 System Workflow

Once a user opens the browser, both policies are read into Identity Armour, and one comprehensive policy is created. When a user visits a website, the web content is loaded and then encountered JavaScript is run by the extension's JavaScript interpreter instead of Firefox's JavaScript engine. While the JavaScript is being parsed, if the externally loaded libraries do not match any of the libraries defined in the combined policy, then it is immediately rejected from the queue of scripts to interpret. If any functions are called which are not within the set of allowed functions specified by the policy, they are then terminated before execution. Finally, throughout this process, sources of information are being tainted and tracked through the system. If any information attempts to leave the user's browser, if be by HTTP request or image request, the user will then be prompted whether or not the information should be sent. The next section describes our evaluation of Identity Armour.

3 Analysis and Evaluation

Our evaluation attempts to answer two questions: how usable Identity Armour is and how secure is Identity Armour against simple and sophisticated attacks.

Experimental Setup. The performance portion of the evaluation was run on Firefox 30.0. All experiments were run on a Dell XPS laptop with an Intel Core i9-9980HK CPU@2.4 GHZ with 16 GB of RAM.

3.1 Performance Evaluation

Our analysis takes Alexa's top 10k websites and records time between page content load and JavaScript interpreted. This system is not compared against a modern browser because we believed the main source of overhead comes from using an interpreter written in an interpreted language. Spidermonkey, Firefox's JavaScript interpreter, being written in C++ [7], just by it's compiled nature is faster than the interpreted language version of a JavaScript engine. For this reason, we compared Identity Armour against a baseline JavaScript interpreter within an extension. By comparing Identity Armour to a JavaScript interpreter, implemented in the same manner, we can see the true cost of Identity Armour while browsing. Around 7% of the Alexa's top 10k websites were not able to be tested because of time-outs or problems with compatibility between the system and the websites being visited.

Table 1 shows Identity Armour has a combined system average JavaScript interpretation time of 6070 ms. There were a number of websites' JavaScript that took minutes to interpret, inflating the combined system average; this is evident by the lesser median and a standard deviation (over fifteen seconds). Next, by comparison Identity Armour on average added 491 ms to the interpretation time of the base JavaScript interpreter. The large difference between the median and mean and the relatively large standard deviation in both experiments show the volatility of JavaScript interpretation.

Table 1. Performance evaluation of Identity Armour vs the baseline JavaScript interpreter across Alexa's top ten thousand websites

	Identity Armour timing (ms)	Base JavaScript interpreter timing (ms)
Average load	6070	5579
Median load	3747	3301
Standard deviation	15228	13418

3.2 Attack Analysis

For the security analysis of the system, we decided to use both custom built malicious scripts and scripts from online resources like metasploit and tech blogs. In the scenarios presented, we assume that a user is visiting a malicious website for the first time, with these exploits intentionally embedded within the page, and only a base policy is used in conjunction with Identity Armour. For this experiment, the policy defines the information that should be taint tracked throughout the system and also defines certain functions that allow dynamic execution of code that should not be allowed to run.

For each exploit, either custom or real-world, we manually confirmed whether Identity Armour could stop malicious code from executing. All of the attacks were thwarted without requiring previous knowledge of the website being visited. To further test Identity Armour, we took the original malicious scripts and obfuscated them to make the behavior of the script unintelligible to the human eye. Since our extension enforces policies at the interpreter level, the same exploits were again defeated because the extension works at a lower root of trust than the served JavaScript. This experiment illustrates how Identity Armour can effectively protect users from information theft.

While Identity Armour adds substantial overhead, we believe the additional security offered by Identity Armour outweighs those costs.

4 Related Work

Given the high-profile nature of web security, there are numerous mechanisms proposed and deployed in academia and industry to protect browsers. We compare our work to the most closely related work, in some cases choosing a representative example for lack of space.

Client-Side Browser Protections: A key design aspect of Identity Armour is that it is purely a client-side protection mechanism. Previous approaches have also been purely client-side such as JSand [1]. JSand is representative of numerous works that use purely client-side enforcement but nevertheless, require the cooperation of web developers to deliver the protection framework and configuration from the server. The main difference between Identity Armour and these systems is that we allow for customizable, client-side policies with no cooperation from the server. We note that our system *does* require active participation on the part of the client, both to acquire policies and to install the protection mechanism. Although this requirement can impact adoption, we believe that future work can automatically craft and distribute policies.

Dataflow For JavaScript: One of the key features of Identity Armour is the use of data flow in order to enforce fine-grained mitigation. Several previous works have focused on detecting and blocking information leaks via dataflow, both statically and dynamically [4–6,12]. The most closely related work to our

own is JSFlow [3], which statically defines a set of disallowed flows and dynamically tracks flows from predefined sources to predefined sinks. JSFlow serves as a foundational step in fine-grained privacy protections. As such, the flow-based portion of Identity Armour is inspired by the JSFlow approach, and we build our technology as an extension to the JSFlow codebase. Identity Armour is not limited to purely dataflow-based protections, and allows a user-configurable set of sources and sinks. Identity Armour includes novel user defined policies that then apply features from tools like CSP.

5 Conclusion

With the uncertain intentions of websites, users can not always be sure whether their information is safe. To combat this, we present Identity Armour, a dynamic taint tracking policy enforcement engine easily addable to the browser. We show that many of the most prevalent web security and privacy threats can be mitigated by empowering users to enforce their own policies. With the an average increase of 8.8% in JavaScript interpretation, we believe that our approach is comparable to other, less-flexible browser protections. Our results show that a variety of attacks, both straight forward and obfuscated, can be stopped. To further expand upon this work, we propose that future researchers find a more effective form of automatic policy construction and create an enforcement engine that can be used on a modern system. Ultimately, we hope that Identity Armour will inspire additional work in user-centric policy creation and enforcement.

References

1. Agten, P., Van Acker, S., Brondsema, Y., Phung, P.H., Desmet, L., Piessens, F.: JSand: complete client-side sandboxing of third-party javascript without browser modifications. In: Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC '12, , New York, NY, USA, pp. 1–10. Association for Computing Machinery (2012)
2. Pew Research Center. Americans complicated feelings about social media in an era of privacy concerns. Accessed 06 Aug 2018
3. Hedin, D., Birgisson, A., Bello, L., Sabelfeld, A. JSFlow: tracking information flow in JavaScript and its APIs. In: Cho, Y., Shin, S.Y., Kim, S.W., Hung, C.C., Hong, J., (eds.) Proceedings of the 29th Symposium on Applied Computing, pp. 1663–1671. ACM (2014)
4. Kashyap, V., et al.: JSAI: a static analysis platform for javascript. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 121–132 (2014)
5. Madsen, M., Livshits, B., Fanning, M.: Practical static analysis of javascript applications in the presence of frameworks and libraries. FSE **2013**, 499–509 (2013)
6. Madsen, M., Møller, A.: Sparse dataflow analysis with pointers and reachability. In: Müller-Olm, M., Seidl, H. (eds.) SAS 2014. LNCS, vol. 8723, pp. 201–218. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10936-7_13
7. MDN Contributors. SpiderMonkey: The Mozilla JavaScript runtime (2019). Accessed 21 June 2020

8. Norberg, P.A., Horne, D.R., Horne, D.A.: The privacy paradox: personal information disclosure intentions versus behaviors. J. Consum. Aff. **41**(1), 100–126 (2007)
9. Roth, S., Barron, T., Calzavara, S., Nikiforakis, N., Stock, B.: Complex security policy? a longitudinal analysis of deployed content security policies (2020)
10. Schwartz, P.M.: Property, privacy, and personal data. Harv. L. Rev. **117**, 2056 (2003)
11. Stanton, B., Theofanos, M.F., Prettyman, S., Furman, S.: Security fatigue. IT Prof. **18**(05), 26–32 (2016)
12. Tripp, O., Ferrara, P., Pistoia, M.: Hybrid security analysis of web javascript code via dynamic partial evaluation. In Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp. 49–59 (2014)